
Dane w Pythonie

Tablice, numpy i inne

1

Wprowadzenie do tablic



Definicja tablicy

Tablice jako struktury danych

Tablice w Pythonie są fundamentalnymi strukturami danych, które umożliwiają przechowywanie i organizowanie elementów w sposób umożliwiający efektywne przetwarzanie oraz dostęp do danych za pomocą indeksów.

Różnice między tablicami a listami

Typy elementów

Tablice przechowują elementy jednego typu, podczas gdy listy mogą zawierać różne typy danych, co zwiększa ich elastyczność.

Rozmiar i zmienność

Tablice mają stały rozmiar, co ogranicza ich elastyczność, natomiast listy mogą dynamicznie zmieniać rozmiar w trakcie działania programu.

Wydajność operacji

Tablice, szczególnie te z NumPy, są bardziej wydajne w operacjach na dużych zbiorach danych w porównaniu do list.

Inna próba definicji

Tablica to struktura danych pozwalająca przechowywać wiele elementów tego samego typu w uporządkowanej formie, najczęściej w postaci wierszy i kolumn. Jest to szczególnie przydatne, gdy dane mają regularną strukturę (np. macierz, tabela).

W Pythonie, chociaż nie ma wbudowanego typu tablicy, możemy tworzyć tablice dwuwymiarowe za pomocą list zagnieżdzonych. Jest to intuicyjny sposób, który daje duże możliwości manipulacji danymi.

Dodatkowo, aby efektywnie operować na dużych zbiorach danych i wykonywać zaawansowane obliczenia, często stosuje się biblioteki takie jak **NumPy**, które wprowadzają zoptymalizowane tablice wielowymiarowe.

Typy danych w tablicach

01

Podstawowe typy danych

W tablicach w Pythonie można — — — — — przechowywać różne podstawowe typy danych, takie jak liczby całkowite, zmiennoprzecinkowe oraz łańcuchy znaków, co umożliwia elastyczne zarządzanie danymi.

02

Złożone struktury danych

Tablice mogą zawierać złożone typy, takie jak listy i słowniki, co pozwala na tworzenie wielowymiarowych struktur danych, ułatwiających organizację i analizę informacji.

2

Tworzenie tablic

Tworzenie tablic

untitled — ~/Documents/Pulsar

```
untitled •  
1 # jak stworzyć tablicę dwuwymiarową  
2  
3  
4 macierz = [  
5     [1, 2, 3],  
6     [4, 5, 6],  
7     [7, 8, 9]  
8 ]  
9  
10 # jak uzyskać dostęp do elementów  
11 print(macierz[0][0]) # 1 (pierwszy wiersz, pierwsza kolumna)  
12 print(macierz[2][1]) # 8 (trzeci wiersz, druga kolumna)
```

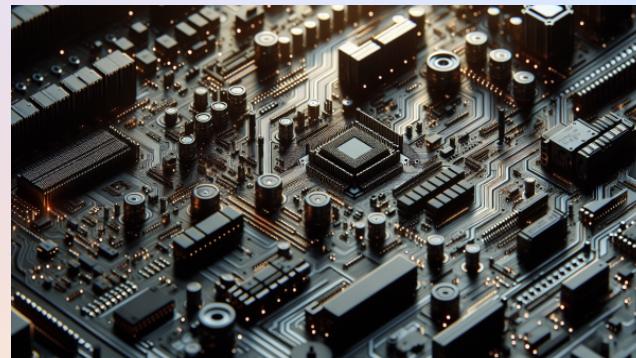
Tworzenie tablicy jednowymiarowej

Definicja i zastosowanie



Tablica jednowymiarowa w Pythonie, reprezentowana jako lista, służy do przechowywania uporządkowanej kolekcji elementów, co ułatwia ich dostęp i manipulację.

Inicjalizacja tablicy



Tworzenie tablicy jednowymiarowej polega na przypisaniu wartości do listy, co można osiągnąć poprzez prostą deklarację, jak w przykładzie z inicjalizacją elementów.

Indeksowanie i modyfikacja



Elementy tablicy jednowymiarowej są dostępne za pomocą indeksów, co umożliwia ich łatwą modyfikację oraz iterację, co jest kluczowe w programowaniu.

Tworzenie tablicy dwuwymiarowej

01



02



03

Inicjalizacja tablicy

Tablicę dwuwymiarową można zainicjować za pomocą list zagnieżdżonych, co pozwala na łatwe tworzenie macierzy o dowolnych wymiarach.

Dynamika rozmiaru

Możliwe jest dynamiczne tworzenie tablicy dwuwymiarowej, co umożliwia dostosowanie liczby wierszy i kolumn w trakcie działania programu.

Zastosowanie pętli

Użycie pętli `for` do iteracji przez wiersze i kolumny tablicy dwuwymiarowej ułatwia przetwarzanie i manipulację danymi w macierzy.

Tworzenie tablic dynamicznych

Dynamiczne zarządzanie pamięcią

Tworzenie dynamicznych tablic w Pythonie pozwala na elastyczne dostosowywanie rozmiaru tablicy w trakcie działania programu, co jest szczególnie przydatne w aplikacjach wymagających zmiennej ilości danych.

Tak po ludzku

Tworzenie tablic dynamicznych polega na tym, że tablica jest tworzona w czasie działania programu, a jej rozmiar lub zawartość może być zmieniana w zależności od potrzeb, zamiast być ustaloną z góry. W Pythonie można to osiągnąć za pomocą list zagnieżdżonych lub funkcji generujących.

Tworzenie tablic dynamicznych

```
untitled •  
1  
2 # Zaczynamy moi drodzy od tworzenia całkowicie pustej tablicy  
3 tablica = []  
4  
5 # Dodajemy wiersze w czasie tworzenia programy – co się nazywa dynamicznym  
6 # tworzeniem tablic  
7 tablica.append([1, 2, 3]) # Dodaj pierwszy wiersz  
8 tablica.append([4, 5, 6]) # Dodaj drugi wiersz  
9 tablica.append([7, 8, 9]) # Dodaj trzeci wiersz  
10  
11 print(tablica)  
12 # Output: [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
```

Tworzenie tablic dynamicznych – dynamicznie można zmieniać treści

```
untitled •  
1  
2 # Dynamiczna zmiana zawartości tablicy:  
3  
4 tablica[1][2] = 99 # Zmiana wartości w drugim wierszu, trzeciej kolumnie  
5 print(tablica)  
6 # Output: [[0, 0, 0, 0], [0, 0, 99, 0], [0, 0, 0, 0]]  
7  
8 # Dodanie nowego wiersza  
9 tablica.append([10, 11, 12, 13])  
10 print(tablica)  
11 # Output: [[0, 0, 0, 0], [0, 0, 99, 0], [0, 0, 0, 0], [10, 11, 12, 13]]
```

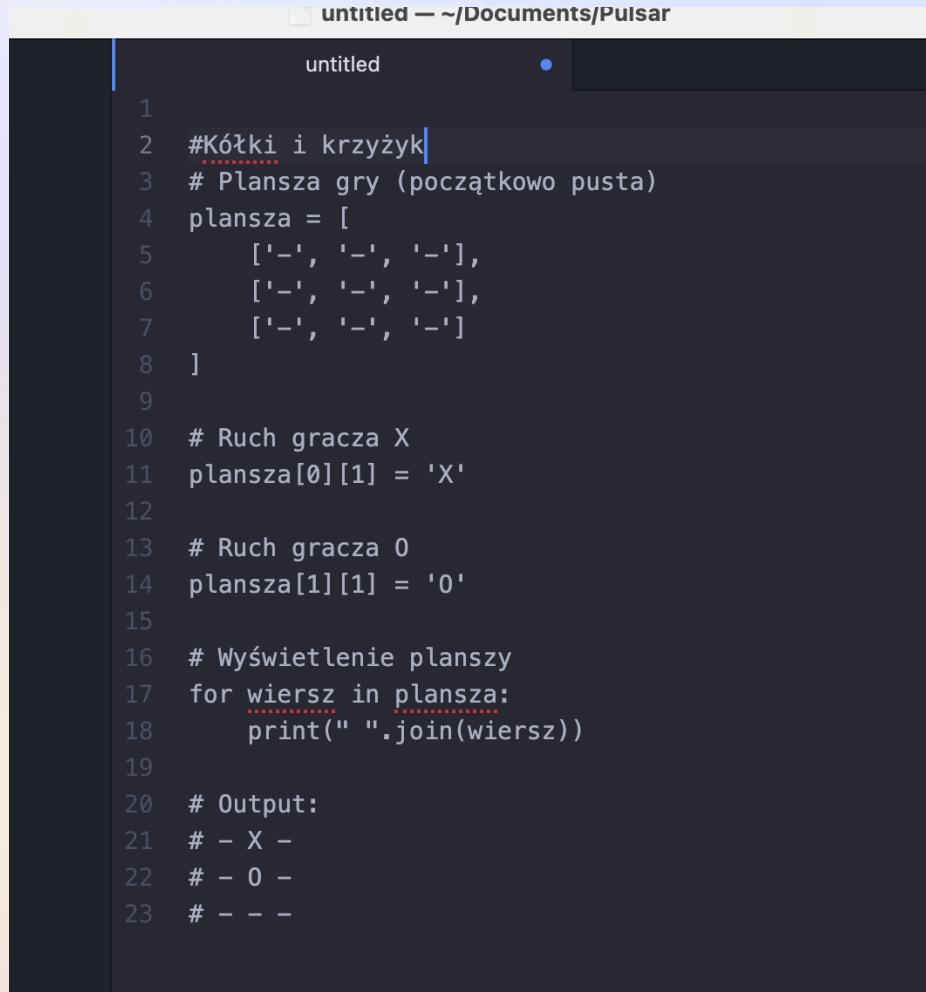
Same tablice też można dynamicznie generować – na przykład poprzez pętle

```
untitled •  
1 wiersze = 3  
2 kolumny = 4  
3  
4 macierz = [[0] * kolumny for _ in range(wiersze)]  
5 print(macierz)  
6 # Output: [[0, 0, 0, 0], [0, 0, 0, 0], [0, 0, 0, 0]]
```

Parę przykładów zastosowania

```
untitled •  
1  
2 # Tablica przechowująca wyniki uczniów z 3 przedmiotów  
3 wyniki = [  
4     [85, 90, 92], # Wyniki ucznia 1  
5     [78, 88, 84], # Wyniki ucznia 2  
6     [92, 95, 96]  # Wyniki ucznia 3  
7 ]  
8  
9 # Dostęp do wyniku ucznia 2 z przedmiotu 1  
10 print("Wynik ucznia 2 z przedmiotu 1:", wyniki[1][0]) # Output: 78  
11  
12 # Średnia wyników ucznia 1  
13 srednia_ucznia1 = sum(wyniki[0]) / len(wyniki[0])  
14 print("Średnia ucznia 1:", srednia_ucznia1) # Output: 89.0
```

Parę przykładów zastosowania



The screenshot shows a code editor window titled "untitled — ~/Documents/Pulsar". The code is a Python script for a simple tic-tac-toe game. It starts with a comment "#Kółki i krzyżyk" followed by a blank line. Then it defines an empty list "plansza" and initializes it with three rows of three dashes each. Next, it performs two moves: placing an 'X' at index [0][1] and an 'O' at index [1][1]. Finally, it prints the board state by iterating over the "plansza" list and joining the elements of each row with spaces.

```
1
2 #Kółki i krzyżyk
3 # Plansza gry (początkowo pusta)
4 plansza = [
5     ['-', '-', '-'],
6     ['-', '-', '-'],
7     ['-', '-', '-']
8 ]
9
10 # Ruch gracza X
11 plansza[0][1] = 'X'
12
13 # Ruch gracza O
14 plansza[1][1] = 'O'
15
16 # Wyświetlenie planszy
17 for wiersz in plansza:
18     print(" ".join(wiersz))
19
20 # Output:
21 # - X -
22 # - O -
23 # - - -
```

Parę przykładów zastosowania

studenci.py ×

```
1 import matplotlib.pyplot as plt
2 # Dane: tablica jako os X i Y
3 x = [1, 2, 3, 4, 5] # Os X
4 y = [2, 4, 6, 8, 10] # Os Y (np. wartości zależne od X)
5 # Tworzenie wykresu
6 plt.plot(*args: x, y, marker='o', label='Funkcja y = 2x')
7 # Dodanie tytułu i opisów osi
8 plt.title('Wykres liniowy')
9 plt.xlabel('Os X')
10 plt.ylabel('Os Y')
11 # Dodanie legendy
12 plt.legend()
13 # Wyświetlenie wykresu
14 plt.show()
```

Parę przykładów zastosowania – od listy do tablicy

The screenshot shows the PyCharm IDE interface. On the left, the code editor displays a Python script named `studenci.py`. The script contains the following code:

```
1 import matplotlib.pyplot as plt
2 # Dane: tablica jako os X i Y
3 x = [1, 2, 3, 4, 5] # Oś X
4 y = [2, 4, 6, 8, 10] # Oś Y
5 # Tworzenie wykresu
6 plt.plot(*args: x, y, marker='o')
7 # Dodanie tytułu i opisów osi
8 plt.title('Wykres liniowy')
9 plt.xlabel('Oś X')
10 plt.ylabel('Oś Y')
11 # Dodanie legendy
12 plt.legend()
13 # Wyświetlenie wykresu
14 plt.show()
```

At the bottom of the code editor, the terminal window shows the command: `/python /Users/przemekz/Documents/Pycharm` and the output: `code 0`.

On the right, the 'Plots' tool window is open, displaying a line plot titled 'Wykres liniowy'. The plot shows a linear function $y = 2x$ with data points at $(1, 2)$, $(2, 4)$, $(3, 6)$, $(4, 8)$, and $(5, 10)$. The x-axis is labeled 'Oś X' and ranges from 1.0 to 5.0. The y-axis is labeled 'Oś Y' and ranges from 2 to 10. A legend indicates the series is 'Funkcja $y = 2x$ '. The plot is a 640x480 PNG (32-bit color) file, 19.9 kB in size.

Parę przykładów zastosowania – od listy do tablicy

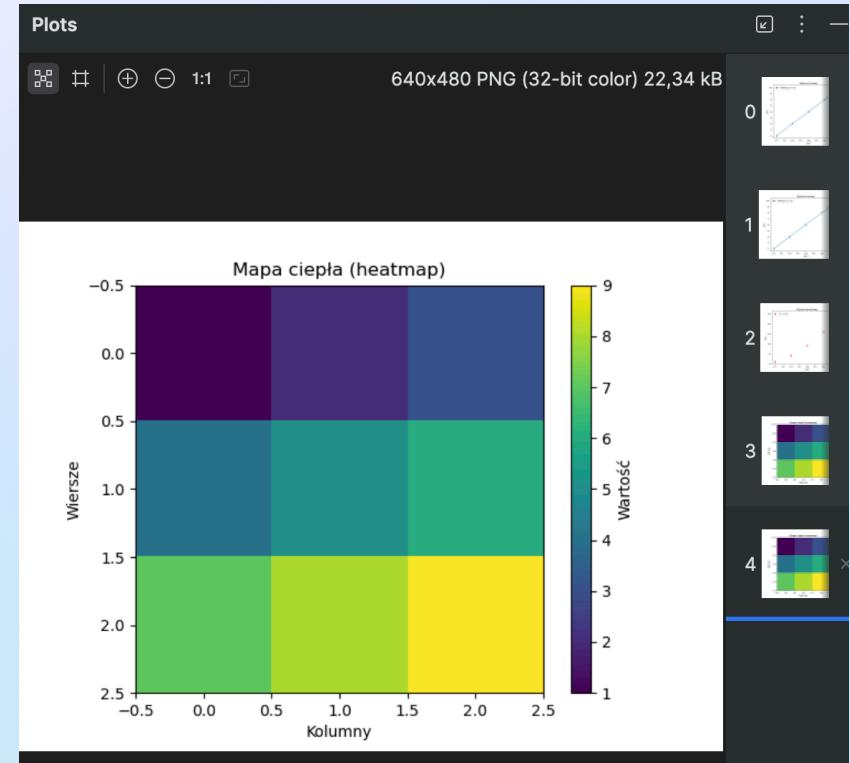
The screenshot shows a Jupyter Notebook environment. On the left, a code cell named "studenci.py" contains the following Python code:

```
1 import matplotlib.pyplot as plt
2
3 # Dane do wykresu
4 x = [1, 2, 3, 4, 5] # Oś X
5 y = [1, 4, 9, 16, 25] # Oś Y (np. kwadraty X)
6
7 # Tworzenie wykresu punktowego
8 plt.scatter(x, y, color='red', label='y = x^2')
9 # Dodanie tytułu i etykiet
10 plt.title('Wykres punktowy')
11 plt.xlabel('Oś X')
12 plt.ylabel('Oś Y')
13 plt.legend()
14 # Wyświetlenie wykresu
15 plt.show()
```

On the right, the "Plots" pane displays three scatter plots. The first plot, titled "Wykres punktowy", shows red dots representing the points (x, y) where $y = x^2$. The x-axis is labeled "Oś X" and ranges from 1.0 to 5.0. The y-axis is labeled "Oś Y" and ranges from 0 to 25. A legend entry "y = x^2" with a red dot is present. The second plot, indexed 1, and the third plot, indexed 2, are smaller versions of the same scatter plot.

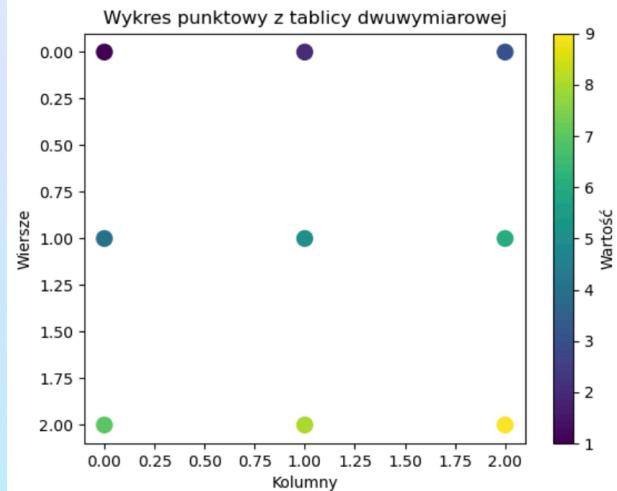
Parę przykładów zastosowania – I same tablice

```
untitled •  
1  
2 import matplotlib.pyplot as plt  
3 import numpy as np  
4  
5 # Tablica dwuwymiarowa (np. wyniki eksperymentów)  
6 tablica = [  
7     [1, 2, 3],  
8     [4, 5, 6],  
9     [7, 8, 9]  
10 ]  
11  
12 # Tworzenie mapy ciepła  
13 plt.imshow(tablica, cmap='viridis', interpolation='nearest')  
14  
15 # Dodanie kolorowej legendy (barwy odpowiadają wartościom)  
16 plt.colorbar(label='Wartość')  
17  
18 # Dodanie tytułu i osi  
19 plt.title('Mapa ciepła (heatmap)')  
20 plt.xlabel('Kolumny')  
21 plt.ylabel('Wiersze')  
22  
23 # Wyświetlenie wykresu  
24 plt.show()
```



Parę przykładów zastosowania – I same tablice

```
1 import matplotlib.pyplot as plt
2
3 # Tablica dwuwymiarowa (np. wyniki pomiarów)
4 tablica = [
5     [1, 2, 3],
6     [4, 5, 6],
7     [7, 8, 9]
8 ]
9
10 # Współrzędne X, Y i wartości
11 x = []
12 y = []
13 wartosci = []
14
15 # Przekształcenie tablicy w listy współrzędnych i wartości
16 for i, wiersz in enumerate(tablica):
17     for j, wartosc in enumerate(wiersz):
18         x.append(j)
19         y.append(i)
20         wartosci.append(wartosc)
21
22 # Rysowanie punktów z kolorami zależnymi od wartości
23 plt.scatter(x, y, c=wartosci, cmap='viridis', s=100)
24
25 # Dodanie legendy i opisów osi
26 plt.colorbar(label='Wartość')
27 plt.title('Wykres punktowy z tablicy dwuwymiarowej')
28 plt.xlabel('Kolumny')
29 plt.ylabel('Wiersze')
30
31 # Odwrócenie osi Y dla lepszego odzwierciedlenia układu wierszy
32 plt.gca().invert_yaxis()
33
34 # Wyświetlenie wykresu
35 plt.show()
```



Przykład z użyciem pętli

01



02



03

Inicjalizacja tablicy dynamicznej

Użycie pętli do tworzenia tablicy dynamicznej pozwala na elastyczne dostosowanie jej rozmiaru w zależności od potrzeb programu.

Modyfikacja wartości w tablicy

Indeksacja umożliwia łatwą aktualizację wartości w tablicy, co jest kluczowe w dynamicznych aplikacjach.

Iteracja przez elementy

Pętle pozwalają na efektywne przetwarzanie wszystkich elementów tablicy, co ułatwia ich analizę i wyświetlanie.

Przykład iteracji:

```
untitled •  
1 macierz = [  
2     [1, 2, 3],  
3     [4, 5, 6],  
4     [7, 8, 9]  
5 ]  
6  
7 for wiersz in macierz:  
8     for element in wiersz:  
9         print(element, end=' ')  
10 # Output: 1 2 3 4 5 6 7 8 9
```

3

Operacje na tablicach

Dostęp do elementów tablicy

Indeksowanie elementów tablicy

Elementy tablicy w Pythonie są dostępne za pomocą indeksów, co umożliwia ich łatwe pozyskiwanie i modyfikację, zarówno w tablicach jednowymiarowych, jak i dwuwymiarowych.

Wykorzystanie pętli do iteracji

Pętle, takie jak `for`, pozwalają na efektywne przetwarzanie wszystkich elementów tablicy, co jest przydatne w analizie danych i ich wyświetlaniu.

Modyfikacja elementów tablicy

Zastosowanie pętli

Pętle, takie jak 'for', umożliwiają masową modyfikację elementów tablicy, co zwiększa efektywność operacji na dużych zbiorach danych.

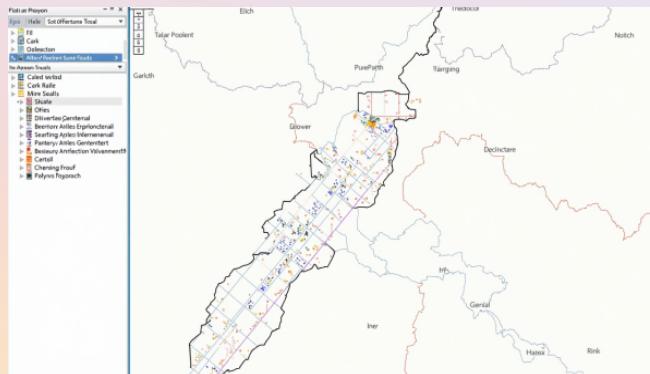
Indeksowanie dla modyfikacji

Modyfikacja elementów tablicy wymaga znajomości indeksów, które w Pythonie zaczynają się od zera, co jest kluczowe dla poprawnych operacji.

Dynamiczne aktualizacje

Modyfikacja elementów tablicy pozwala na dynamiczne dostosowywanie danych, co jest istotne w aplikacjach wymagających bieżącej aktualizacji informacji.

Iteracja przez tablicę



Podstawowe techniki iteracji

Iteracja przez tablicę dwuwymiarową w Pythonie najczęściej wykorzystuje podwójne pętle `for`, co pozwala na efektywne przetwarzanie każdego elementu w macierzy.

$$\begin{aligned} &= (2l_{1d})t_{2d} = \frac{f\ell}{11,2^2} = i_d = 2t \cdot 1 \cdot (1) \\ &= + (H_{i_4} 2teg. \times = \lambda_{i_1, 0})h + (t_{i_4} = (5t).h. \\ &= (di, \lambda = \left(\frac{f\ell_{i_2} (ikd)}{0tn_d} \right)^3 \\ &= l_{fegl} = (2l_4)^2 \end{aligned}$$

Zastosowanie funkcji `enumerate`

Funkcja `enumerate` umożliwia jednoczesne uzyskanie indeksów wierszy i kolumn, co ułatwia śledzenie pozycji elementów podczas iteracji przez tablicę.

```
(fue sapt: = 300;
not: fixt( = tin confrec of = 4;
coertion oalds; = 17;
dentieren acienne = )T4; 16)
outere: 1:2,
outifen 3)
fiej al.(1x1] + 11;
ce(ven( = 1s:30)
57: 58:
```

Praktyczne zastosowania iteracji

Iteracja przez tablicę pozwala na realizację różnych operacji, takich jak obliczanie sumy elementów czy wyszukiwanie konkretnych wartości, co zwiększa funkcjonalność kodu.

Iteracja przez tablicę

```
untitled •  
1 owoce = ['jabłko', 'banan', 'pomarańcza']  
2  
3 for indeks, owoc in enumerate(owoce):  
4     print(f'Indeks: {indeks}, Owoc: {owoc}')  
5  
6 # Output:  
7 # Indeks: 0, Owoc: jabłko  
8 # Indeks: 1, Owoc: banan  
9 # Indeks: 2, Owoc: pomarańcza
```

Zastosowanie funkcji `enumerate`

Funkcja enumerate służy do iterowania po elementach iterowalnego obiektu, jednocześnie zwracając ich indeksy, co ułatwia dostęp zarówno do pozycji, jak i wartości elementu w jednej pętli.

Wyszukiwanie elementów w tablicy

Wyszukiwanie z użyciem NumPy

W bibliotece NumPy, funkcja `np.where` umożliwia szybkie i efektywne wyszukiwanie elementów w tablicach, zwracając indeksy spełniające określone warunki, co znacząco przyspiesza operacje na dużych zbiorach danych.

```
[[0 0]
 [1 0]
 [2 0]
 [0 1]
 [1 1]
 [0 1]
 [0 2]
 [1 2]
 [2 2]
 [0 1]]
```

Result:

```
array([3 5 9], dtype=int64)
```

4

Wprowadzenie do NumPy

Co to jest NumPy?

01

02

03

Podstawowe zastosowania

NumPy jest wykorzystywane w obliczeniach naukowych, analizie danych oraz w uczeniu maszynowym, co czyni je kluczowym narzędziem w tych dziedzinach.

Wydajność obliczeniowa

Dzięki optymalizacji operacji na tablicach, NumPy znacząco przyspiesza obliczenia w porównaniu do standardowych struktur danych w Pythonie.

Wsparcie dla bibliotek

NumPy stanowi fundament dla wielu popularnych bibliotek Pythona, takich jak Pandas i Matplotlib, co zwiększa jego znaczenie w ekosystemie Pythona.

Instalacja i import NumPy

Kroki instalacji NumPy

Użyj komendy `pip install numpy` w terminalu, aby zainstalować bibliotekę NumPy oraz jej zależności w systemie operacyjnym.

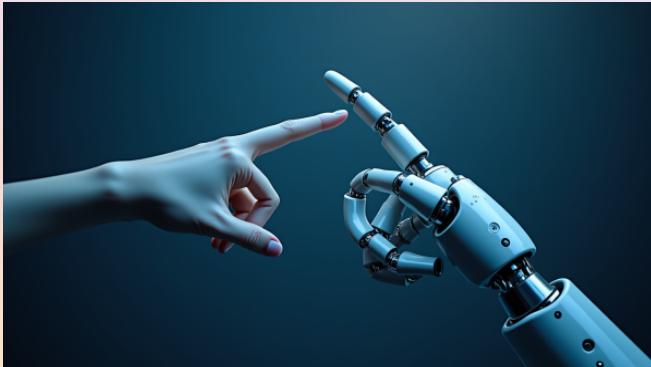
Weryfikacja instalacji

Sprawdź poprawność instalacji, uruchamiając `pip show numpy`, co wyświetli szczegóły zainstalowanej wersji biblioteki.

Importowanie w Pythonie

Użyj `import numpy as np`, aby zimportować NumPy, co umożliwia łatwe korzystanie z funkcji biblioteki w kodzie.

Tworzenie tablic NumPy



Inicjalizacja tablicy z listy

Tablice NumPy można łatwo tworzyć z istniejących list Pythona, co pozwala na szybkie przekształcenie danych w formę tablicy, idealną do obliczeń numerycznych.



Tworzenie tablic o stałych wartościach

Funkcje `numpy.zeros()` i `numpy.ones()` umożliwiają szybkie generowanie tablic wypełnionych zerami lub jedynkami, co jest przydatne w wielu zastosowaniach obliczeniowych.



Generowanie tablic z zakresu

Użycie `numpy.arange()` pozwala na tworzenie tablic z sekwencjami liczb w określonym zakresie, co jest szczególnie przydatne w analizie danych i symulacjach.

5

Operacje na tablicach NumPy

Podstawowe zastosowania numpy

```
studenci.py ×

1 import numpy as np
2
3 macierz = np.array([
4     [1, 2, 3],
5     [4, 5, 6],
6     [7, 8, 9]
7 ])
8 print(macierz)
```

• Dlaczego używać NumPy?

- Optymalizacja: szybsze operacje niż na listach wbudowanych.
- Wsparcie dla operacji matematycznych i naukowych.

Przykład operacji matematycznych

```
studenci.py ×

1 import numpy as np
2
3 macierz = np.array([
4     [1, 2, 3],
5     [4, 5, 6],
6     [7, 8, 9]
7 ])
8
9 print(macierz + 10)
10 # Output: [[11 12 13] [14 15 16] [17 18 19]]
```



Indeksowanie i wycinanie

Zaawansowane techniki wycinania

NumPy umożliwia wycinanie tablic na podstawie warunków logicznych, co pozwala na selekcję elementów spełniających określone kryteria, co jest kluczowe w analizie danych i optymalizacji obliczeń.

Przykład indeksowania

```
studenci.py ×  
1 import numpy as np  
2  
3 # Tablica 2D (3x3)  
4 tablica = np.array([  
5     [10, 20, 30],  
6     [40, 50, 60],  
7     [70, 80, 90]  
8 ])  
9  
10 # Dostęp do elementu w wierszu 2, kolumnie 3  
11 print(tablica[1, 2]) # Output: 60  
12  
13 # Dostęp do elementu w wierszu 1, kolumnie 1  
14 print(tablica[0, 0]) # Output: 10
```

Przykład wycinania

```
studenci.py ×

1 tablica = np.array([
2     [10, 20, 30],
3     [40, 50, 60],
4     [70, 80, 90]
5 ])
6 # Wybranie pierwszych dwóch wierszy i dwóch pierwszych kolumn
7 podtablica = tablica[:2, :2]
8 print(podtablica)
9 # Output:
10 # [[10 20]
11 #  [40 50]]
12 # Wybranie ostatniego wiersza
13 ostatni_wiersz = tablica[-1, :]
14 print(ostatni_wiersz) # Output: [70 80 90]
```

Modyfikacja tablic NumPy

01

Zmiana wartości elementów

Użycie indeksowania pozwala na precyzyjne modyfikowanie wartości w tablicach NumPy, co jest kluczowe w analizie danych.

02

Dodawanie nowych elementów

Funkcje takie jak `np.append()` umożliwiają dodawanie nowych wierszy lub kolumn, co zwiększa elastyczność tablic.

03

Usuwanie elementów

Funkcja `np.delete()` pozwala na usunięcie wybranych elementów z tablicy, co jest istotne w zarządzaniu danymi.

Section 6

Zaawansowane funkcje NumPy

Reshape i transpozycja tablic

Zasady reshaping

Nowe wymiary muszą być zgodne z oryginalnym rozmiarem tablicy, co zapewnia integralność danych podczas przekształcania.

Automatyczne dopasowanie

Użycie `-1` w `reshape` pozwala na automatyczne obliczenie wymiaru, co upraszcza proces dostosowywania tablic.

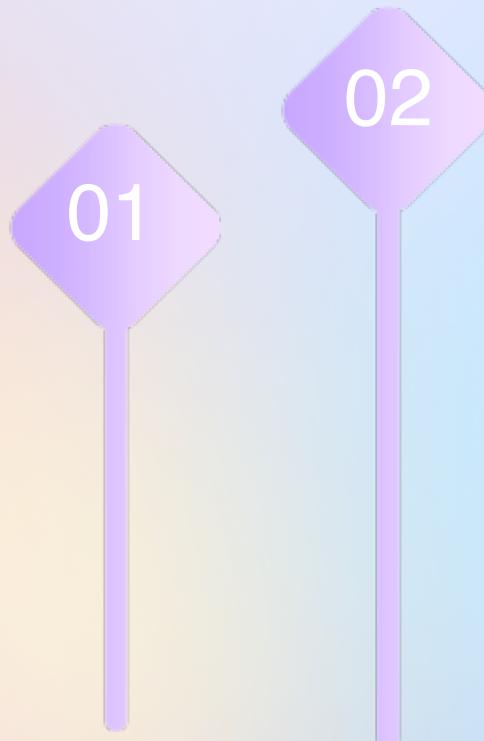
Praktyczne zastosowania

Reshape i transpozycja są kluczowe w analizie danych, umożliwiając dostosowanie formatu danych do wymagań algorytmów uczenia maszynowego.

Łączanie i dzielenie tablic

Techniki łączenia tablic

W Pythonie można łączyć tablice za pomocą operatora `+`, metody `extend()` oraz funkcji `itertools.chain()`, co pozwala na elastyczne zarządzanie danymi.



Metody dzielenia tablic

Dzielenie tablic można realizować poprzez indeksowanie i wycinanie, a także za pomocą wyrażeń listowych, co umożliwia selekcję elementów na podstawie określonych warunków.

Broadcasting w NumPy

Mechanizm dopasowania kształtów



Broadcasting w NumPy automatycznie dostosowuje kształty tablic, co pozwala na wykonywanie operacji matematycznych na tablicach o różnych wymiarach bez konieczności ręcznego przekształcania.

Zastosowanie w analizie danych



Dzięki broadcasting, analitycy danych mogą szybko i efektywnie przeprowadzać operacje na dużych zbiorach danych, co znacząco przyspiesza proces analizy i obliczeń.

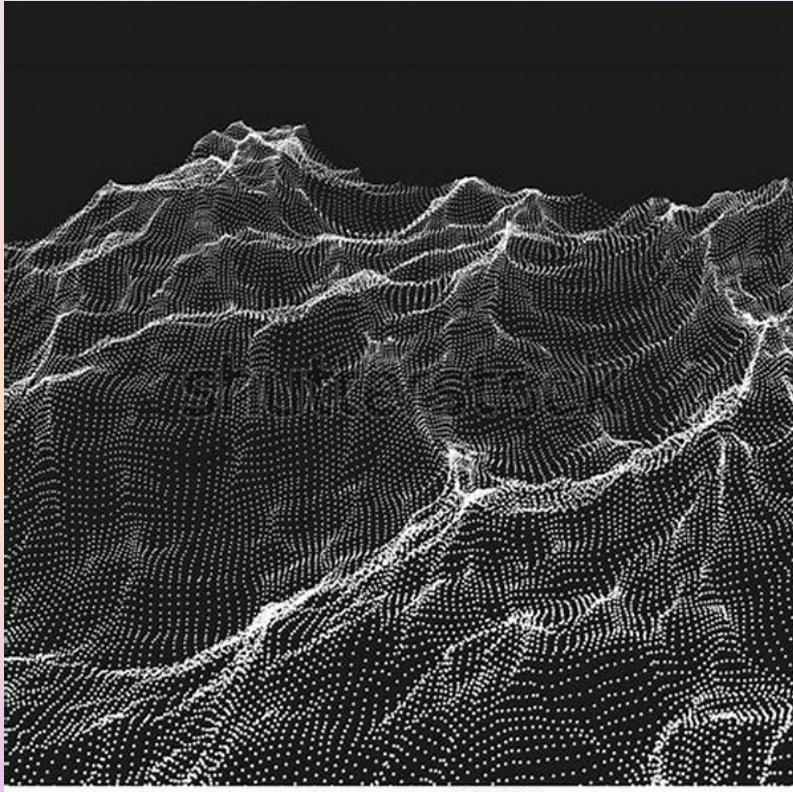
Przykłady praktyczne



Typowe zastosowania broadcasting obejmują dodawanie skalarów do tablic, dodawanie wektorów do macierzy oraz mnożenie tablic, co ilustruje jego wszechstronność w obliczeniach.

Section 7

Przykłady praktyczne



Tworzenie i modyfikacja tablicy 3x3

Dynamiczne zmiany w tablicy

W Pythonie, tablica 3x3 może być dynamicznie modyfikowana, co pozwala na elastyczne dostosowywanie danych w aplikacjach, umożliwiając łatwe dodawanie, usuwanie lub aktualizację elementów w czasie rzeczywistym.

Obliczanie sumy elementów tablicy

Suma w tablicach jednowymiarowych

```
Alve te2o  
rel, iinlt = rid, fnicks 0,  
0t niled iolog;  
of 1 16x _lennment;  
Seat 0,  
mdsping 4;  
Ild; tist a]  
silV;  
az;
```

Aby obliczyć sumę elementów w tablicy jednowymiarowej, można użyć funkcji `sum()`, co zapewnia prostotę i efektywność w obliczeniach, eliminując potrzebę pętli.

Suma w tablicach dwuwymiarowych

1050	654
1051	105601
2015	2103
01	0141
2015	2216
10615	
£1014 506	£15
£27257196	262
5151	
41060	
0 £104	
11-157	1506 100
£E 21b	2047052 261

W przypadku tablic dwuwymiarowych, suma elementów wymaga zagnieżdżonej pętli lub funkcji `numpy.sum()`, co umożliwia szybkie obliczenia w macierzach.

Zastosowanie NumPy

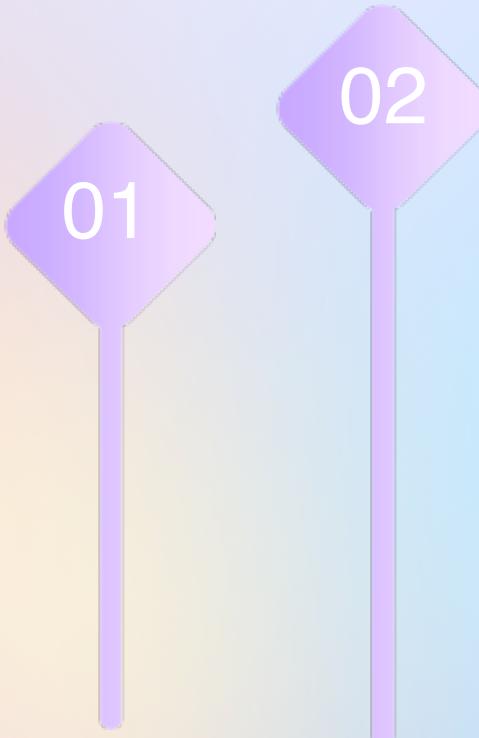


Użycie biblioteki NumPy do obliczania sumy elementów w tablicach pozwala na optymalizację operacji, co jest szczególnie korzystne przy pracy z dużymi zbiorami danych.

Przykład z transpozycją tablicy NumPy

Zastosowanie transpozycji

Transpozycja macierzy w NumPy jest kluczowa w analizie danych, umożliwiając przekształcanie danych z formatu kolumnowego na wierszowy, co ułatwia dalsze operacje analityczne.



Wydajność operacji

Użycie funkcji `np.transpose()` w NumPy jest zoptymalizowane, co pozwala na szybkie i efektywne przekształcanie dużych zbiorów danych bez znacznego obciążenia pamięci.

Section 8

Zastosowania tablic w praktyce

Analiza danych z użyciem tablic

01

Efektywność operacji matematycznych

Tablice NumPy umożliwiają szybkie wykonywanie operacji matematycznych na dużych zbiorach danych, co znaczowo przyspiesza proces analizy.

02

Zastosowanie w statystyce

Tablice są kluczowe w obliczeniach statystycznych, umożliwiając łatwe obliczanie miar takich jak średnia, mediana i odchylenie standardowe.

03

Integracja z innymi bibliotekami

Tablice NumPy są fundamentem dla wielu bibliotek analitycznych, takich jak Pandas, co zwiększa ich użyteczność w analizie danych.

Obliczenia naukowe z NumPy

Wydajność obliczeń

NumPy przyspiesza obliczenia dzięki wektoryzacji, co pozwala na równoległe przetwarzanie danych i minimalizację czasu wykonania operacji matematycznych.

Wsparcie dla macierzy

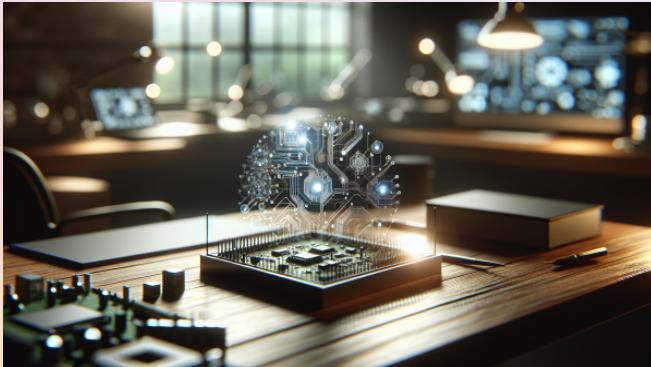
NumPy obsługuje operacje na macierzach, co jest kluczowe w obliczeniach naukowych, umożliwiając łatwe wykonywanie złożonych operacji liniowych.

Funkcje statystyczne

NumPy oferuje wbudowane funkcje statystyczne, takie jak `np.mean()` i `np.std()`, które ułatwiają analizę danych i obliczenia statystyczne.

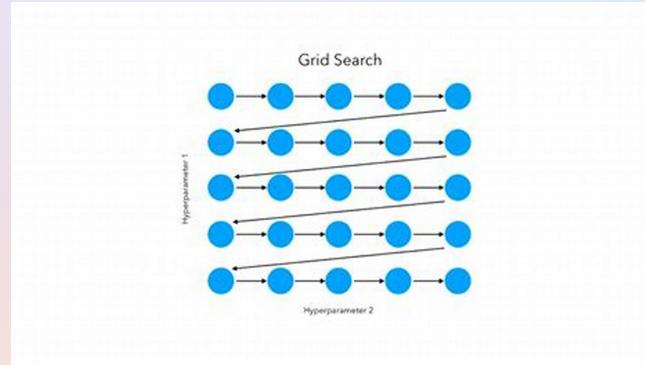


Przykłady zastosowań w machine learning



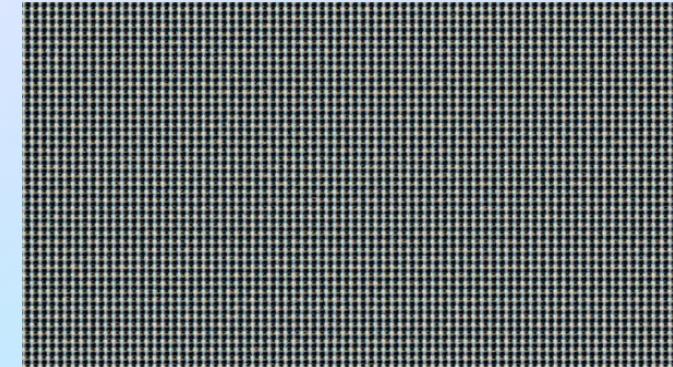
Zastosowanie w klasyfikacji

Tablice NumPy są wykorzystywane do reprezentacji danych wejściowych w modelach klasyfikacyjnych, takich jak SVM czy drzewa decyzyjne, co umożliwia efektywne przetwarzanie i analizę danych.



Optymalizacja hiperparametrów

W machine learning, tablice NumPy wspierają proces optymalizacji hiperparametrów, umożliwiając przeprowadzanie eksperymentów z różnymi wartościami w celu poprawy wydajności modelu.



Analiza wyników modelu

Tablice NumPy ułatwiają analizę wyników modeli machine learning, pozwalając na szybkie obliczenia metryk, takich jak dokładność czy F1-score, co jest kluczowe w ocenie efektywności algorytmu.

Section 9

Podsumowanie i pytania

Kluczowe wnioski

01

Fundamentalna rola tablic

Tablice w Pythonie są kluczowe dla organizacji danych, umożliwiając efektywne przechowywanie i manipulację dużymi zbiorami informacji.

02

Zastosowanie NumPy

Biblioteka NumPy znaczowo zwiększa wydajność obliczeń numerycznych, oferując zaawansowane funkcje do pracy z tablicami.

03

Praktyczne umiejętności

Umiejętność tworzenia i modyfikacji tablic jest niezbędna w programowaniu, szczególnie w kontekście analizy danych i obliczeń naukowych.

Pytania i odpowiedzi

Inicjalizacja tablicy dwuwymiarowej

Można użyć list zagnieźdzonych lub biblioteki NumPy, co pozwala na efektywne tworzenie macierzy o różnych wymiarach.

Różnice między tablicami a listami

Tablice w Pythonie są jednorodne, co oznacza, że przechowują elementy tego samego typu, w przeciwieństwie do elastycznych list.



Iteracja przez tablicę

Użycie zagnieźdzonych pętli `for` umożliwia przetwarzanie każdego elementu tablicy dwuwymiarowej, co jest kluczowe w analizie danych.

Dalsze kroki w nauce Pythona

Zaawansowane techniki programowania

Zgłębianie programowania obiektowego, dekoratorów oraz obsługi wyjątków w Pythonie pozwala na tworzenie bardziej złożonych i efektywnych aplikacji, co zwiększa umiejętności programistyczne.

Praktyczne projekty i wyzwania

Realizacja projektów, takich jak analiza danych czy budowa modeli machine learning, umożliwia zastosowanie zdobytej wiedzy w praktyce, co przyspiesza rozwój umiejętności programowania.

Thank You

Contact:popai@example.com